2

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 89-549
June 1989

Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms

Thang Bui, Christopher Heigham, Curt Jones, and Tom Leighton

Abstract

In this paper, we compare the performance of two popular graph bisection algorithms. We also present an empirical study of a new heuristic, first proposed in Bui, Chaudhuri, Leighton, Sipser [BCLS87], that dramatically improves the performance of these bisection algorithms on graphs with small (≤ 4) average degree.

In the graph bisection problem we are given a graph $G = (V,E)$ and are asked to partition the vertex set V into two equal-sized subsets $V1$ and $V2$ in such a way that the size of the cut between them is minimized. The *cut* of $V1$ and $V2$ is the number of edges with one endpoint in $V1$ and the other endpoint in $V2$. The minimum cut over all bisections is known as the *bisection width* of the graph. Graph bisection has applications in VLSI placement and routing problems. The problem is known to be NP-hard.

Bisecting graphs is one problem domain where simulated annealing Johnson, Aragon, MeGeoch, Schevon [JCAMS84] has been used with some success. Kernighan-Lin [KL70] is the recognized champion among the classical approaches to the graph bisection problem. Unfortunately it is known to fail badly on certain types of graphs (e.g., the ladder graph).

In this paper both methods for solving the graph bisection problem with be outlined. Results of applying the algorithms to this problem domain will be given. Also a new heuristic, called compaction, which we originally proposed in [BCLS87] is shown to dramatically improve the performance of both approaches on graphs with small average degree.

Microsystems Research Center Room 39-321 | Massachusetts Institute of Technology | Cambridge Massachusetts 02139 | Telephone (617) 253-8138

Acknowledgements

To appear in the *Proceedings, Design Automation Conference*, June 1989. This research was supported in part by the Defense Advanced Research Projects Agency under contract number N00014-87-K-0825, DARPA/U.S. Army Research Office under contract DAAL 03-86-K-0171, Air Force Contract AFOSR-86-0076, and NSF Presidential Young Investigator Award.

Author Information

Bui and Jones: Computer Science Department, Pennsylvania State University, University Park, PA 16802.

Heigham and Leighton: Department of Mathematics and Laboratory for Computer Science, MIT, Room NE43-332, Cambridge, MA 02139. (617) 253-5876.

Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms

Thang Bui*, Christopher Heigham**, Curt Jones* and Tom Leighton**†

*Computer Science Department
Pennsylvania State University
University Park, Pennsylvania 16802

**Math Department & Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

In this paper, we compare the performance of two popular graph bisection algorithms. We also present an empirical study of a new heuristic, first proposed in [BCLS87], that dramatically improves the performance of these bisection algorithms on graphs with small (≤ 4) average degree.

I. Introduction

In the graph bisection problem we are given a graph $G = (V, E)$ and are asked to partition the vertex set V into two equal-sized subsets V_1 and V_2 in such a way that the size of the cut between them is minimized. The *cut* of V_1 and V_2 is the number of edges with one endpoint in V_1 and the other endpoint in V_2. The minimum cut over all bisections is known as the *bisection width* of the graph. Graph bisection has applications in VLSI placement and routing problems. The problem is known to be NP-hard.

Bisecting graphs is one problem domain where simulated annealing [JCAMS84] has been used with some success. Kernighan-Lin [KL70] is the recognized champion among the classical approaches to the graph bisection problem. Unfortunately it is known to fail badly on certain types of graphs (e.g., the ladder graph).

In this paper both methods for solving the graph bisection problem will be outlined. Results of applying the algorithms to this problem domain will be given. Also a new heuristic, called compaction, which we originally proposed in [BCLS87] is shown to dramatically improve the performance of both approaches on graphs with small average degree.

II. The Simulated Annealing Algorithm

Simulated annealing was proposed by Kirkpatrick, Gelatt and Vecchi [KGV83], as a different approach for approximating solutions to difficult combinatorial optimization problems. The method is based on ideas

†Research supported in part by DARPA Contract N0014-87-K-825, Airforce Contract OSR-86-0076, Army Contract DAAL-03-86-K-0171, and NSF Presidential Young Investigator Award.

from statistical mechanics and motivated by the analogy to the behavior of a physical system in the presence of a heat bath. The computer scientist can view the process as a modification of the iterative improvement (or "neighborhood search") technique

In iterative improvement an initial solution is repeatedly improved by making small local changes until no such alteration yields a better solution. The "goodness" of a solution is based on a cost function that is to be minimized. Local knowledge (small local changes) is used to decrease the cost function until no further reduction is possible. The drawback with this type of search is the problem of stopping at a local, but not global, optimum. The process is usually carried out several times with different randomly generated starting configurations to deal with this problem.

Statistical mechanics deals with properties of the large numbers of atoms to be found in samples of solid or liquid matter. The behavior of a system in thermal equilibrium at a given temperature is observed by experiments. One aspect of a system studied here is what happens to the substance in the limit of low temperature. The atoms may remain fluid or solidify, and if they solidify they may form a crystalline solid or a glass.

Experiments that determine the low temperature state of a substance are done by careful annealing, first applying heat to the substance, then lowering the temperature slowly, and spending a long time at temperatures in the vicinity of the freezing point. If the cooling is too rapid the substance may get out of equilibrium, resulting in a metastable, locally optimal condition.

Iterative improvement is much like the processes modeled by statistical mechanics, with the cost function playing the role of energy. Using only rearrangements that lower the cost function of the system is like extremely rapid quenching *from high temperature* to zero. The results are similarly, sometimes only metastable, local optimal solutions. Simulated annealing provides a generalization in which controlled uphill steps can also be incorporated in the search for a better solution. These uphill steps keep the solution from cooling too rapidly.

The generic simulated annealing approach is shown in Figure 1.

```
GET INITIAL SOLUTION S
GET INITIAL TEMPERATURE T
WHILE (NOT YET FROZEN) DO
BEGIN
  WHILE (NOT YET IN EQUILIBRIUM) DO
  BEGIN
   PICK A RANDOM SOLUTION S'
   LET Δ = CHANGE IN COST
   IF Δ ≤ 0 SET S = S'
   ELSE SET S = S' WITH PROBABILITY e^(-Δ/T)
  END
  REDUCE TEMPERATURE
END
OUTPUT SOLUTION S
```

Figure 1. Generic simulated annealing algorithm.

Controlled uphill movements are allowed by step 10 of the procedure given in Figure 1. According to Kirkpatrick et al. [KGV83],

> "This step allows gross features of the eventual state of the system to appear at high temperature, the details develop at lower temperatures."

This algorithm is based on a simple procedure introduced by Metropolist et al. [MRRTT53] to simulate a collection of atoms in equilibrium at a given temperature. At each step an atom is given a small random displacement and the resulting change, ΔE, in energy is computed. If $\Delta E \leq 0$, the displacement is accepted. The other situation, $\Delta E > 0$, is treated probabilistically. The displacement is accepted with probability equal to $e^{(-\Delta E/T)}$ **. The generic simulated annealing algorithm extends this procedure by using a cost function instead of energy and letting the temperature be a parameter that is reduced as the process continues.

III. The Kernighan-Lin Algorithm

One well known method for solving the graph bisection problem is the Kernighan-Lin heuristic. It seems to work well in practice and its variations are some of the most widely used graph bisection algorithms. The algorithm starts with an arbitrary bisection, say (A, B), and improves upon it. The algorithm interchanges subsets $X \subset A$, $Y \subset B$ and $|X| = |Y| < |A|$ such that the size of the bisection is decreased. The method selects elements of X and Y sequentially.

We will need some additional notation to fully describe this algorithm. Let $G = (V, E)$ be a graph on $2n$ vertices and let (A, B) be a bisection of G. Denote the cardinality of the bisection (A, B) by $|(A, B)|$. For each vertex $a \in A$, we define the gain, g_a, of a as the difference between the number of edges connecting a to vertices in B and the number of edges connecting a to vertices in A. We extend this definition to pairs of vertices, one in A and one in B. We denote by $g_{a,b}$ the reduction in the size of the bisection when a and b are interchange. Clearly,

$$g_{a,b} = g_a + g_b - 2\delta(a, b)$$

where

$$\delta(a, b) = \begin{cases} 1, & \text{if } (a, b) \in E; \\ 0, & \text{otherwise.} \end{cases}$$

The algorithm first computes $g_{a,b}$ for all $a \in A$, $b \in B$. It then chooses $a_1 \in A$, $b_1 \in B$ such that

$$g_{a_1,b_1} = max\{g_{a,b} | a \in A,\ b \in B\}$$

The algorithm then updates all the gains for each vertex in V with respect to the bisection that would be created by interchanging a_1 and b_1. The algorithm then repeats this process but does not consider the vertices just exchanged again during this pass. The process is repeated until all vertices have been considered. We now have a list of n pairs of vertices. If all these pairs were interchanged the size of the bisection would not change. The algorithm picks a $k < n$ such that the interchange of the first k pairs of vertices will give a maximum reduction in the size of the bisection over all choices of k. This whole process makes up one pass of the algorithm. The procedure may have a fixed number of passes or it can run until no improvement is possible. The algorithm is listed in Figure 2. For a more detailed treatment of this and other bisection algorithms see [Bui86].

** Strictly speaking we need a Boltzmann factor, k_B, multipling T in this equation [KGV83].

Begin

1. Compute g_a, g_b for each $a \in A$, $b \in B$.
2. $Q_A = \emptyset$, $Q_B = \emptyset$.
3. for $i = 1$ to $n-1$ do

   Begin

4. Choose $a_i \in A - Q_A$ and $b_i \in B - Q_B$ such that g_{a_i,b_i} is maximal over all choices of a and b.
5. Set $Q_a = Q_a \cup \{a_i\}, Q_B = Q_B \cup \{b_i\}$
6. for each $a \in A - Q_A$ do

   $g_a = g_a + 2\delta(a, a_i) - 2\delta(a, b_i)$

7. for each $b \in B - Q_B$ do

   $g_b = g_b + 2\delta(b, a_i) - 2\delta(b, a_i)$

   End

9. Choose $k \in \{1, \ldots, n-1\}$ to maximize $\sum_{i=1}^{k} g_{a_i,b_i}$
10. Interchange the subsets $\{a_1, \ldots, a_k\}$ and $\{b_1, \ldots, b_k\}$ to get a new bisection.

End

Figure 2. One pass of the Kernighan-Lin graph bisection algorithm.

IV. Graph Models

We used three random graph models, $\mathcal{G}_{breg}(2n, b, d)$, $\mathcal{G}_{2set}(2n, p_A, p_B, bis)$ and $\mathcal{G}_{NP}(2n, p)$ to test the bisection algorithms. The graph model $\mathcal{G}_{NP}(2n, p)$ contains all simple graphs on $2n$ vertices, in which an edge between any two vertices is present with probability p, independent of any other edge. The expected average degree of each vertex is $(2n-1)p$ (a binomial distribution). This model was used in [JAMS84], however it can be shown that the graphs in $\mathcal{G}_{NP}(2n, p)$, for a fixed p, have a large minimal cut. The minimal cut usually contains about half the number of edges in the graph. Hence a random partition will differ only slightly from the optimal partition. Thus, this model may not distinguish good heuristics from mediocre ones.

The model $\mathcal{G}_{2set}(2n, p_A, p_B, bis)$ differs from $\mathcal{G}_{NP}(2n, p)$ slightly. First the graph is broken into two sets (A and B) each of size n. Edges are placed between any two vertices in set A with probability p_A. Edges are also placed between any two vertices in set B with probability p_B. Then exactly bis edges are randomly placed between vertices of sets A and B. This puts an upper bound (bis) on the bisection of the graph. All edges are placed independently of other edges. With this model it is difficult to generate graphs with small average degree and still be confident that the optimal bisection is equal to bis. Many times graphs with small average degree (< 4) and large expected bisection ($> \log 2n$) generated with this model have minimum bisection width much smaller than the expected bisection width. If the average degree of the graph is less than two, then the graphs generated under this model usually have an optimal bisection width of zero. This makes obtaining meaningful results with this graph model difficult unless the average degree of the graph is large. In order to remedy these two problems a graph model is needed that allows one to specify with high

probability the exact bisection width as well as yield a uniform placement of the edges. One such graph model is $\mathcal{G}_{breg}(2n, b, d)$.

The graph model $\mathcal{G}_{breg}(2n, b, d)$ was introduced by Bui et al., in [BCLS87]. This class of graphs consists of all simple regular graphs with $2n$ nodes, where each node has degree d and the graph has bisection width b. For graphs generated under this model the minimum bisection is much smaller than the average bisection so this graph model overcomes the weakness of $\mathcal{G}_{NP}(2n, p)$. Also with this model we were able to construct with high probability graphs of small degree with a unique small bisection. This is an improvement over the model $\mathcal{G}_{2set}(2n, p_A, p_B, bis)$. Most of the graphs used to test our implementations were generated with this model, we did however test the effectiveness of the algorithms on graphs generated using the other random graph models as well as using grid graphs, ladder graphs, and binary trees.

V. Compaction

It was noticed in [BCLS87] that both Kernighan-Lin and simulated annealing worked much better on graphs when the graph has a high (> 3) average degree. On sparse graphs the algorithms ran longer with less than optimal results. Kernighan-Lin also performs better when given a good starting bisection. Goldberg and Burstein [GB83] also discovered that Kernighan-Lin based algorithms did better on networks of large degree. Previous experimental results in [BCLS87] showed that a new heuristic (compaction) improved the performance of the Kernighan-lin algorithm. The compaction heuristic combines two nodes of a graph G into one node of a graph G'. We wanted to see if the technique would produce similar improvements for simulated annealing and also determine how all four methods compared.

Bisection using compaction works on a graph $G = (V, E)$ as follows:

1. Form a maximum random matching M of the graph G.
2. Form a new graph G' by contracting the edges in the random matching M. That is coalesce the two endpoints of an edge in the random matching M to form a new vertex. All vertices incident to the two original vertices are now incident to the new vertex just formed.
3. Run the bisection heuristic on G' to obtain the bisection (A', B').
4. Uncompact the edges to obtain the original graph and create a initial bisection (A, B) from (A', B').
5. Use (A, B) as the starting configuration for the bisection procedure on the original graph.

This method will cause the average degree of the graph G' to be larger than the average degree of G. If the bisection algorithm finds a good bisection of G' then hopefully the corresponding initial bisection (A, B) of G will be close to an optimal bisection of G. We shall denote the methods resulting from using compaction as compacted simulated annealing (CSA) and compacted Kernighan-Lin (CKL).

VI. Results

We generated 556 random graphs to study how well the algorithms perform. For each graph we ran each procedure from two different randomly generated initial bisections. All bisection results reported here will be based on the best solution of the two trials for that graph. All timing results will be the total time it took the procedure to complete both starting configurations (including the time to generate the initial bisections). The graphs ranged from five hundred to five thousand vertices. The bisection widths ranged

from a cut size of zero to $\sqrt{|V|}$. For each setting of the parameters under the model $\mathcal{G}_{breg}(2n, b, d)$ we generated three different random graphs. Under the other graph models one graph was generated with each setting of the parameters. Most of the random graphs (368) were generated using the model $\mathcal{G}_{breg}(2n, b, d)$. Of the remaining random graphs, 132 were generated using the model $\mathcal{G}_{2set}(2n, p_A, p_B, bis)$, and 56 under the model $\mathcal{G}_{NP}(2n, p)$. A collection of special graphs (grid, ladder and binary trees) were also used for testing.

In these tests both methods performed very well in terms of the bisection returned by the algorithm when the average degree of the graph was greater than three. The compaction heuristic provided improvements on graphs with average degree of two to three. In these graphs compaction decreased the size of the bisection found and in many cases actually ran faster than the standard algorithms. For graphs with average degree less than two both the standard and compacted versions of the algorithms usually found a bisection of cost zero. The Kernighan-Lin implementation was faster than the simulated annealing procedure.

It should be noted that under the model $\mathcal{G}_{breg}(2n, b, d)$ graphs of degree two must consist only of a collection of cordless cycles. As such the optimal bisection is ≤ 2 for all settings of b. This collection of simple cycles does give some indication of how well the algorithms perform on very sparse graphs. However one could just use a depth first search algorithm to obtain a better approximation or one could solve the problem exactly in time $O(n^2)$ for these graphs. The rest of this section will be presented as a series of observations followed by data to support the observation being made.

Observation 1: The bisection algorithms improve as the average degree increases.

Both algorithms do much better on graphs of degree 4 than graphs of degree 3. (Tests were also performed on graphs with degree larger than 4, but results on graphs of degree 4 exemplify what happens for graph of higher degree). This is most evident in the tests run on larger graphs. In particular on graphs from $\mathcal{G}_{breg}(5000, b, 3)$ both algorithms without compaction usually found bisections that were twenty to fifty times larger than the expected bisections. But on graphs from $\mathcal{G}_{breg}(5000, b, 4)$ the expected bisection was always found. Also, the algorithms usually ran faster on regular degree 4 graphs than regular degree three graphs. On graphs of 5,000 vertices the algorithms ran up to three times faster for the Kernighan-Lin algorithm and almost two times faster for the simulated annealing algorithm. This is because it takes fewer passes for the algorithms to converge on degree 4 graphs. In [BCLS87] it was conjectured that large degree graphs may have very few local optimums (bisections close to the optimal bisection). This would help explain our data.

Observation 2: Compaction improves performance on small degree graphs both in the time needed by the algorithms and in the quality of the solution returned.

From the tables in the appendix it can be seen that compaction improves the performance of Kernighan-Lin more than simulated annealing. In the appendix one can also see the dramatic improvement compaction provides the algorithms on graphs from $\mathcal{G}_{breg}(5000, b, 3)$. Due to lack of space we have not shown the data for graphs of size smaller than 2000 but we observed that once again compaction provides more of a benefit as the graph size increases. In graphs from $\mathcal{G}_{breg}(5000, b, 3)$ the smallest improvement compaction provided was over 90 percent. Similar significant improvements are also observed for graphs in $\mathcal{G}_{2set}(5000, p_A, p_B, b)$. Notice also that compaction usually sped up the Kernighan-Lin algorithm as well as improved the quality of the solution returned. Compaction did not slowdown simulated annealing significantly on graphs of degree

≤ 3. Compacted Kernighan-Lin was three times faster than the standard Kernighan-Lin algorithm and ten times faster than simulated annealing on graphs from $\mathcal{G}_{breg}(5000, b, 3)$. On graphs of degree 4 compaction did not cause a significant slowdown for Kernighan-Lin while returning the expected bisection.

Observation 3: Compaction also helps on some special graphs

We also tested the two bisection heuristics on some special classes of graphs to see how much improvement compaction would make. The graphs ranged in size from 100 to 5,000 vertices. In Table 1 we show the average improvement in the cut size of the bisection returned that compaction made for each of the two procedures. In the appendix we also give more detailed tables for these special graphs.

Graph type	Avg compaction improvement over KL	Avg compaction improvement over SA
Grid	13%	34%
Ladder	12%	24%
Binary Tree	56%	17%

Table 1. Bisection width improvement made by compaction. Best of two starts.

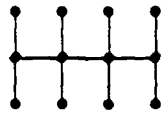

Figure 3. *An example of a ladder graph*

Observation 4: Without compaction Kernighan-Lin algorithm runs faster and produces better solutions than simuated annealing. On binary trees and ladder graphs, however, Kernighan-Lin algorithm did not do as well as simulated annealing.

In our tests the Kernighan-Lin algorithm was a much faster procedure. On large graphs the simulated annealing procedure took up to twenty times longer to converge to a solution. Simulated annealing does allow for a tradeoff between the quality of solution and time used to find the solution. The "fine tuning"of the annealing schedule can be a big job, as we found out. When the annealing procedure did terminate quickly it was usually at a far from optimal solution.

In the quality of the solution returned, the Kernighan-Lin procedure was more consistent than simulated annealing. In our test we started each procedure from two different initial configurations. Simulated annealing occasionally showed large differences in the results of the two trials. As the average degree of the graph increased, both algorithms started to find the expected bisections. On graphs of average degree of 2.5 to 3.5, when a noticeable difference was observed in the quality of the bisection returned, the Kernighan-Lin procedure had the better bisection sixty percent of the time. Simulated annealing did out perform Kernighan-Lin on binary trees, and ladder graphs.

Observation 5: With compaction, simulated annealing is still slower than Kernighan-Lin algorithm but there is no big difference in the quality of the solutions.

Compaction definitely helped both algorithms. Simulated annealing was still a much slower procedure. When there is a difference in the quality of the solutions by Kernighan-Lin and simulated annealing the former did return slightly better bisections, the exceptions being on binary trees and ladder graphs. Compacted simulated annealing found smaller cuts than compacted Kernighan-Lin on these graphs.

VII. Conclusion

We have shown further empirical evidence that Kernighan-Lin and simulated annealing perform better as the average degree of the graph increases. This provided the basis for the compaction heuristic which drastically improved the performance on small degree graphs. The performance increase was in the quality of the solution found and for Kernighan-Lin the speed in which they were obtained. Our empirical data suggest that the compaction heuristic should be used on graphs with average degree of four or less.

We also found Kernighan-Lin to be a much faster algorithm than simulated annealing on the graphs that we tested. Part of this is due to the fact that simulated annealing must run until the temperature cools down to a freezing point. Simulated annealing may then continue to search for an optimal solution a long time after finding a good bisection. In fact simulated annealing may migrate away from an optimal solution if it is found at a high temperature. One must then save the best bisection found as the algorithm progresses. This increases both the time and storage requirements of the algorithm. This is why compaction does not increase the time needed by Kernighan-Lin to complete but may increase the time needed by simulated annealing to return a bisection. Attempts at correcting this flaw caused the algorithm to terminate prematurely. This gave us some indication of how much work was needed to "fine tune"the simulated annealing algorithm. One may have to spend a great deal of computation time to find the correct setting of the parameters for a particular class of problems.

VIII. References

[Bui86] T. Bui, "Graph Bisection Algorithms", Ph.D. Thesis, Dept. of Electrical Enginerring and Computer Science, Massachusetts Institute of Technology, 1986.

[BCLS87] T. Bui, S. Chaudhuri, F. Leighton, and M. Sipser, "Graph Bisection Algorithms with Good Average Case Behavior", Combinatorica 7 (2) 1987, pp. 171–191.

[GB83] M. Goldberg and M. Burstein, "Heuristic Improvement Technique for Bisection of VLSI Networks", Proceedings of the IEEE International conference on computer design: VLSI in computers (ICCD '83), 1983, pp. 122–125.

[JCAMS84] D. Johnson, C. Aragon, L. MeGeoch and C. Schevon, Unpublished Manuscript 1984.

[KL70] B. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", The Bell System Tech J., Vol. 49, No. 2, Feb. 1970, pp. 291–307.

[KGV83] S. Kirkpatrick, C. Gelatt Jr., and M. Vecchi, "Optimization by simulated annealing", Science 220 May 13, 1983, pp. 671–680.

[MRRTT53] N. Metropolis, A. Rosenbluth, M Rosenbluth, A. Teller, and E. Teller, "Equations of State Calcu-

lations by Fast Computing Machines," Journal Chemical Phys. 21. pp. 1087–1092, 1953.

IX. Appendix

In the following tables we have summarized the nontrivial ($2n \geq 2000$) data from our tests. The columns of the tables show the expected bisection width (b) of the graph along with the cut size of the bisection returned by the standard (b_{SA}, b_{KL}) and compacted (b_{CSA}, b_{CKL}) versions of each algorithm. The time needed to compute the bisection is listed under the cut size. We also show for each algorithm the relative improvement compaction provides both in the cut size of the bisection returned and the time needed to find that bisection. The cut size improvement is given as a percentage as is the time improvement. The relative time improvement was calculated as follows.

$$t_{woc} = \text{time without compaction}$$

$$t_{wc} = \text{time with compaction}$$

$$\text{Rel. speed up} = \frac{t_{woc} - t_{wc}}{t_{woc}} \times 100$$

The timing results are in terms of cpu minutes. The top line of each row in the following tables represents cut size and the bottom line represents cpu time. Each entry in the table for the $\mathcal{G}_{breg}(2n, b, d)$ model is the average of three different random graphs generated with the same parameters but different random number generator seeds. All random numbers were generated by a Fibonacci random number generator. The programs were coded in C and run on a Vax 780 under BSD Unix Version 4.3.

Ladder graphs

Ladder graph with 3N nodes						
3N	b_{SA} Time	b_{CSA} Time	$\frac{b_{SA}-b_{CSA}}{b_{SA}} \times 100$ Rel. speed up (%)	b_{KL} Time	b_{CKL} Time	$\frac{b_{KL}-b_{CKL}}{b_{KL}} \times 100$ Rel. speed up (%)
300	7 0.46	9 0.76	−29 −65	17 0.03	12 0.05	29 −66
600	24 1.10	15 1.17	38 −6	25 0.05	34 0.07	−36 −40
1200	46 2.89	28 4.03	39 −39	73 0.16	50 0.20	32 −25
1500	50 3.97	42 5.32	16 −28	72 0.15	76 0.18	−6 −20
2100	75 4.88	60 7.45	20 −53	108 0.41	108 0.26	0 37
3000	113 9.34	73 13.84	35 −48	160 0.49	123 0.85	23 −73
4500	160 12.35	122 19.71	24 −60	245 0.67	233 0.60	5 −10
6000	212 18.47	122 27.15	24 −47	245 1.81	233 1.94	5 −7

Grid graphs

N × N grid graph						
N	b_{SA} Time	b_{CSA} Time	$\frac{b_{SA}-b_{CSA}}{b_{SA}} \times 100$ Rel. speed up (%)	b_{KL} Time	b_{CKL} Time	$\frac{b_{KL}-b_{CKL}}{b_{KL}} \times 100$ Rel. speed up (%)
10	14 0.13	10 0.11	29 15	10 0.01	10 0.02	0 −50
20	50 0.66	32 0.96	36 −45	25 0.05	20 0.07	20 −40
30	112 2.94	58 2.68	48 9	37 0.23	30 0.24	20 −4
40	152 3.23	98 3.43	36 −6	55 0.55	40 0.41	27 25
50	246 5.19	194 7.94	21 −53	59 1.11	50 0.76	15 35
60	326 4.29	222 12.43	32 −189	69 2.38	73 1.35	−5 43

Binary trees

Binary tree with N nodes						
N	b_{SA} Time	b_{CSA} Time	$\frac{b_{SA}-b_{CSA}}{b_{SA}} \times 100$ Rel. speed up (%)	b_{KL} Time	b_{CKL} Time	$\frac{b_{KL}-b_{CKL}}{b_{KL}} \times 100$ Rel. speed up (%)
100	3 0.49	4 0.97	−33 −95	8 0.01	6 0.02	25 −98
200	5 1.80	4 2.00	20 −12	17 0.02	10 0.03	41 −56
400	16 3.65	14 6.43	20 −76	29 0.05	13 0.07	55 −45
500	18 3.88	15 5.84	17 −51	34 0.05	10 0.10	71 −95
700	26 7.30	21 9.74	19 −33	41 0.16	19 0.15	54 3
1000	31 12.15	24 15.92	23 −31	74 0.19	32 0.18	57 4
1500	53 18.46	46 22.65	13 −23	103 0.30	37 0.33	64 −11
2000	69 23.02	49 37.77	29 −64	144 0.54	52 0.55	64 −2
3000	112 35.52	72 39.92	36 −12	226 0.72	82 0.79	64 −9
4000	127 51.29	101 77.50	20 −51	304 0.82	108 1.22	64 −49
5000	167 64.94	126 78.73	25 −21	363 1.62	140 1.59	61 2

5000 vertex graphs

$\mathcal{G}_{2set}(5000, p_A, p_B, b)$ with average degree 2.5						
b	b_{SA} Time	b_{CSA} Time	$\frac{b_{SA}-b_{CSA}}{b_{SA}} \times 100$ Rel. speed up (%)	b_{KL} Time	b_{CKL} Time	$\frac{b_{KL}-b_{CKL}}{b_{KL}} \times 100$ Rel. speed up (%)
0	568 30.24	40 46.24	93 −53	485 2.06	2 1.58	100 23
2	611 29.63	1 47.02	100 −59	58 3.68	4 1.48	93 60
4	493 32.07	20 43.63	96 −36	39 2.48	8 1.57	79 37
6	504 29.70	33 45.24	93 −52	90 2.01	18 1.72	80 15
8	616 32.01	28 45.87	95 −43	586 2.45	14 1.54	98 37
22	551 31.25	64 45.64	88 −46	82 3.12	21 2.04	74 34
70	580 28.21	77 45.54	87 −61	613 2.43	55 1.74	91 28
$\mathcal{G}_{2set}(5000, p_A, p_B, b)$ with average degree 3.0						
0	257 32.06	0 47.16	100 −47	13 1.86	3 1.59	77 15
2	423 30.85	3 44.39	99 −44	15 1.86	2 1.39	87 25
4	751 16.41	5 37.35	99 −128	22 1.48	5 1.61	77 −9
6	85 32.26	7 47.30	92 −47	14 2.08	7 1.60	50 23
8	904 32.95	13 46.66	99 −42	24 1.28	8 1.60	67 −25
22	48 33.54	29 46.46	40 −39	33 1.78	24 1.70	27 5
70	870 32.84	70 47.10	92 −43	95 1.69	64 1.60	33 5

$\mathcal{G}_{2set}(5000, p_A, p_B, b)$ with average degree 3.5						
b	b_{SA} Time	b_{CSA} Time	$\frac{b_{SA}-b_{CSA}}{b_{SA}} \times 100$ Rel. speed up (%)	b_{KL} Time	b_{CKL} Time	$\frac{b_{KL}-b_{CKL}}{b_{KL}} \times 100$ Rel. speed up (%)
0	9 33.12	0 48.99	100 −48	6 1.21	0 1.59	100 −31
2	6 34.58	1 48.19	83 −39	7 1.12	1 1.68	86 −50
4	4 35.31	4 47.71	0 −35	6 1.33	5 1.71	17 −28
6	10 34.70	6 49.34	40 −42	8 1.12	5 1.82	38 −63
8	7 31.18	8 50.34	−14 −61	13 1.23	6 1.76	54 −43
22	21 36.21	25 47.29	−19 −31	30 1.12	21 1.84	30 −64
70	74 16.03	67 47.28	9 −195	75 1.33	63 1.84	16 −38
$\mathcal{G}_{2set}(5000, p_A, p_B, b)$ with average degree 4.0						
0	0 34.63	0 47.59	— −37	0 1.07	0 2.49	— −133
2	2 30.80	2 51.24	0 −66	3 1.08	2 1.96	33 −82
4	6 32.71	4 36.25	33 −11	5 1.19	4 2.02	20 −70
6	7 28.66	5 51.24	29 −79	5 1.18	5 2.44	0 −106
8	10 31.72	8 53.00	20 −67	12 1.07	8 2.19	33 −105
22	27 36.21	22 47.58	19 -51	21 1.16	21 2.12	0 −82
70	70 32.20	69 49.94	1 −55	68 1.07	68 2.22	0 −107

$\mathcal{G}_{NP}(5000, p)$**						
Avg Deg	b_{SA} Time	b_{CSA} Time	$\frac{b_{SA}-b_{CSA}}{b_{SA}} \times 100$ Rel. speed up (%)	b_{KL} Time	b_{CKL} Time	$\frac{b_{KL}-b_{CKL}}{b_{KL}} \times 100$ Rel. speed up (%)
2.5	635 34.63	591 47.59	7 −37	641 1.07	557 2.49	13 −133
3.0	809 35.35	764 46.01	6 −30	836 2.23	743 2.47	11 −11
3.5	1227 31.19	1175 47.58	4 −53	1258 2.40	1150 3.03	9 −26
4.0	1672 32.17	1624 48.82	3 −52	1699 2.77	1614 3.63	5 −31

**Each entry is the average of seven random graphs.

$\mathcal{G}_{breg}(5000, b, 3)$						
b	b_{SA} Time	b_{CSA} Time	$\frac{b_{SA}-b_{CSA}}{b_{SA}} \times 100$ Rel. speed up (%)	b_{KL} Time	b_{CKL} Time	$\frac{b_{KL}-b_{CKL}}{b_{KL}} \times 100$ Rel. speed up (%)
0	549 16.32	0 12.50	100 24	836 4.45	0 0.90	100 80
2	740 15.49	2 10.99	99 29	883 3.74	2 1.11	99 70
4	523 15.79	4 14.94	99 5	595 4.80	4 1.14	99 76
6	408 12.71	6 13.09	99 −3	593 4.90	6 1.17	99 76
8	554 15.22	8 14.85	99 2	315 5.83	8 1.24	97 79
12	671 16.22	14 14.5	98 11	595 5.08	12 1.14	98 78
16	727 12.85	17 14.10	98 −10	601 4.65	16 1.15	97 75
20	764 14.68	20 16.66	97 −14	603 7.66	20 1.14	97 85
70	695 15.90	70 17.68	90 −11	891 4.89	72 1.23	92 75
$\mathcal{G}_{breg}(5000, b, 4)$						
0	0 10.0	0 8.93	— 11	0 1.17	0 1.02	— 13
2	2 8.0	2 11.37	0 −42	2 1.17	2 1.14	0 3
4	4 12.09	4 10.51	0 13	4 1.17	4 1.18	0 −1
6	6 8.58	6 10.18	0 −19	6 1.17	6 1.25	0 −7
8	8 9.84	8 11.85	0 −20	8 1.17	8 1.27	0 −9
12	12 8.52	12 8.88	0 −4	12 1.21	12 1.25	0 −3
16	16 9.93	16 8.84	0 11	16 1.22	16 1.27	0 −4
20	20 8.63	20 11.39	0 −32	20 1.17	20 1.22	0 −4
70	70 8.72	70 10.36	0 −19	72 1.17	72 1.35	0 −15

2000 vertex graphs

$\mathcal{G}_{2set}(2000, p_A, p_B, b)$ with average degree 2.5						
b	b_{SA} Time	b_{CSA} Time	$\frac{b_{SA}-b_{CSA}}{b_{SA}} \times 100$ Rel. speed up (%)	b_{KL} Time	b_{CKL} Time	$\frac{b_{KL}-b_{CKL}}{b_{KL}} \times 100$ Rel. speed up (%)
0	63 15.08	17 25.90	73 −72	14 1.05	0 0.55	100 47
2	149 15.56	4 22.67	97 −46	33 1.47	5 0.57	85 61
4	129 16.29	4 24.24	97 −49	33 1.15	5 0.61	85 47
6	252 16.32	6 23.73	98 −45	261 0.53	7 0.60	97 −15
8	224 16.26	8 24.68	96 −52	248 0.76	7 0.55	97 27
20	207 16.41	16 22.85	92 -39	53 0.58	17 0.65	68 −12
44	228 17.47	41 25.06	82 −43	83 0.80	34 0.74	59 7
$\mathcal{G}_{2set}(2000, p_A, p_B, b)$ with average degree 3.0						
0	28 19.13	4 24.48	86 −28	10 0.72	0 0.74	100 −2
2	11 18.21	2 24.83	82 −36	4 0.68	2 0.56	50 18
4	35 18.91	7 25.19	80 −33	12 0.70	4 0.74	67 −4
6	66 18.17	8 27.01	88 −49	10 0.68	5 0.74	50 −9
8	10 17.74	7 25.77	30 −45	10 0.62	7 0.44	30 28
20	30 18.38	24 23.45	20 −28	23 0.61	20 0.60	13 1
44	327 16.45	38 25.92	88 −58	42 0.82	36 0.72	14 13

$\mathcal{G}_{2set}(2000, p_A, p_B, b)$ with average degree 3.5						
b	b_{SA} Time	b_{CSA} Time	$\frac{b_{SA}-b_{CSA}}{b_{SA}} \times 100$ Rel. speed up (%)	b_{KL} Time	b_{CKL} Time	$\frac{b_{KL}-b_{CKL}}{b_{KL}} \times 100$ Rel. speed up (%)
0	0 14.99	0 27.72	— −85	0 0.48	0 0.62	— −29
2	5 17.28	3 25.49	40 −48	3 0.52	2 0.67	33 −29
4	4 17.53	4 26.43	0 −51	4 0.63	4 0.64	0 −2
6	6 17.15	7 26.77	−17 −56	6 0.66	6 0.62	0 6
8	11 15.38	8 24.94	27 −62	9 0.57	8 0.64	11 −12
20	18 15.34	18 26.03	0 −70	17 0.57	16 0.70	6 −23
44	42 18.79	41 26.95	2 −43	41 0.66	41 0.67	0 −1
$\mathcal{G}_{2set}(2000, p_A, p_B, b)$ with average degree 4.0						
0	1 18.53	0 26.91	100 −45	0 0.47	0 0.61	— −30
2	2 15.96	2 27.45	0 −72	7 0.47	2 0.64	71 −36
4	3 17.15	3 25.75	0 −50	3 0.58	3 0.68	0 −18
6	5 15.79	5 25.06	0 −59	5 0.52	5 0.82	0 −57
8	9 16.60	8 28.09	11 −69	10 0.47	8 0.74	20 −59
20	20 16.66	20 26.99	0 −62	20 0.46	20 0.76	0 −64
44	41 15.03	41 27.29	0 −82	41 0.52	41 0.76	0 −46

$\mathcal{G}_{NP}(2000, p)^{**}$						
Avg Deg	b_{SA} Time	b_{CSA} Time	$\frac{b_{SA}-b_{CSA}}{b_{SA}} \times 100$ Rel. speed up (%)	b_{KL} Time	b_{CKL} Time	$\frac{b_{KL}-b_{CKL}}{b_{KL}} \times 100$ Rel. speed up (%)
2.5	245 16.64	223 23.82	9 −103	249 0.74	216 0.76	13 −3
3.0	364 17.42	352 24.92	3 −43	380 0.77	337 0.92	11 −2
3.5	493 18.62	482 25.32	2 −36	510 0.87	468 1.06	6 −2
4.0	658 12.54	633 18.30	4 −46	675 0.60	634 0.91	6 −52

** Each entry is the average of seven random graphs.

$\mathcal{G}_{breg}(2000, b, 3)$						
b	b_{SA} Time	b_{CSA} Time	$\frac{b_{SA}-b_{CSA}}{b_{SA}} \times 100$ Rel. speed up (%)	b_{KL} Time	b_{CKL} Time	$\frac{b_{KL}-b_{CKL}}{b_{KL}} \times 100$ Rel. speed up (%)
0	96 5.52	0 4.03	100 27	0 1.43	0 0.33	— 77
2	2 4.86	2 5.45	0 −12	105 0.81	2 0.40	98 51
4	97 5.05	4 5.13	96 −2	7 1.03	4 0.43	43 58
6	279 5.55	7 4.10	97 26	366 0.62	6 0.42	98 32
8	77 5.41	8 4.48	90 17	10 1.57	8 0.43	20 73
12	159 5.38	12 4.74	92 19	130 1.06	12 0.43	91 59
16	251 5.50	16 4.08	94 26	253 1.05	16 0.44	94 57
20	295 5.51	20 5.07	93 8	221 1.08	20 0.46	91 57
44	291 5.31	57 6.49	80 −22	263 0.72	44 0.48	83 33
$\mathcal{G}_{breg}(2000, b, 4)$						
0	0 2.66	0 3.64	— −37	0 0.46	0 0.41	— 11
2	2 2.66	2 2.97	0 −12	2 0.46	2 0.41	0 11
4	4 2.86	4 3.36	0 −17	4 0.44	4 0.43	0 2
6	6 3.16	6 3.01	0 5	6 0.42	6 0.45	0 −7
8	8 2.92	8 2.80	0 4	8 0.44	8 0.45	0 −2
12	12 3.09	12 3.37	0 −9	12 0.42	12 0.45	0 −7
16	16 3.26	16 4.19	0 −29	16 0.44	16 0.46	0 −5
20	20 3.66	20 2.93	0 20	20 0.48	20 0.49	0 −2
44	44 3.40	57 3.27	−30 4	44 0.48	44 0.49	0 −2